Fachprojekt „Systemsoftwaretechnik“ (SST)

A0 - Der Linux-Kern

SoSe 24 – 09. April 2024

`https://sys.cs.tu-dortmund.de/de/lehre/ss24/sst`

Alexander Lochmann

Lehrstuhl für Informatik 12 – Arbeitsgruppe Systemsoftware
Technische Universität Dortmund

(Mit Material von dem Lehrstuhl 4 der aus Erlangen, der Arbeitsgruppe *SRA* aus Hannover und Arbeitsgruppe *BOSS* aus Bochum)

technische universität
dortmund

Arbeitsgruppe
Systemsoftware

Agenda

Warum eigentlich Linux?

Warum eigentlich Linux?

- aktive und schnelle Entwicklung

  ⇒ riesige Entwicklungsgemeinschaft
- Vielseitigkeit & Portabilität

  ⇒ Smart-TVs, Thermostate, E-Reader, Infotainment, ...
- Skalierbarkeit

  ⇒ Linux für Raspberry Pi bis Supercomputer

Warum eigentlich Linux?

- aktive und schnelle Entwicklung

  ⇒ riesige Entwicklungsgemeinschaft
- Vielseitigkeit & Portabilität

  ⇒ Smart-TVs, Thermostate, E-Reader, Infotainment, ...
- Skalierbarkeit

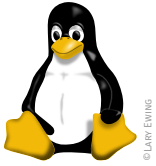
  ⇒ Linux für Raspberry Pi bis Supercomputer
- aktive Forschung

  ⇒ Linux als Forschungsgegenstand und Plattform
- Performance & Effizienz

  ⇒ 100% der Top500 Supercomputer Linux-basiert
- Verbreitung [1, 2, 18]

  ⇒ ca. 80% der Webserver, ca. 70% der Smartphones

Warum eigentlich Linux?



- aktive und schnelle Entwicklung
  ⇒ riesige Entwicklungsgemeinschaft
- Vielseitigkeit & Portabilität
  ⇒ Smart TVs, Thermostate, E-Reader, Infotainment, ...
- Skalierbarkeit
  ⇒ Linux für Raspberry Pi bis Supercomputer
- aktive Forschung
  ⇒ Linux als Forschungsgegenstand und Plattform
- Performance & Effizienz
  ⇒ 100% der Top500 Supercomputer Linux-basiert
- Verbreitung [1, 2, 18]
  ⇒ ca. 80% der Webserver, ca. 70% der Smartphones

Vor allem die **Quelloffenheit** macht Linux zu einem der wichtigsten Betriebssysteme der Zukunft!

Was ist eigentlich Linux? [13, 16]

Linux Kernel

= essentielle OS-Funktionalität

(Prozess- und Speicherverwaltung, Gerätetreiber, …)

Was ist eigentlich Linux? [13, 16]

Linux Kernel
= essentielle OS-Funktionalität

(Prozess- und Speicherverwaltung, Gerätetreiber, …)

Betriebssystem
= Sammlung an Programmen zur Systemnutzung

(Linux Kernel, Nutzerverwaltung, GNU Werkzeuge, …)

Was ist eigentlich Linux? [13, 16]

Linux Kernel
= essentielle OS-Funktionalität

(Prozess- und Speicherverwaltung, Gerätetreiber, …)

Betriebssystem
= Sammlung an Programmen zur Systemnutzung

(Linux Kernel, Nutzerverwaltung, GNU Werkzeuge, …)

Distribution
= Betriebssystem und Dienste / Utilities

(Paketmanagement, Anwendungsprogramme wie Browser und Editoren, …)

Was ist eigentlich Linux? [13, 16]

Linux Kernel
= essentielle OS-Funktionalität

(Prozess- und Speicherverwaltung, Gerätetreiber, …)

Betriebssystem
= Sammlung an Programmen zur Systemnutzung

(Linux Kernel, Nutzerverwaltung, GNU Werkzeuge, …)

Distribution
= Betriebssystem und Dienste / Utilities

(Paketmanagement, Anwendungsprogramme wie Browser und Editoren, …)

Linux Kernel

- **Eckdaten**
  - ursprünglich 1991 von Linus Torvalds entwickelt
  - veröffentlicht unter der GPLv2 [3]

- **Struktur** [4, 12, 14]
  - = dynamischer, monolithischer Kernel
  - dynamisch ladbare Module
  - (logische) Aufteilung in Subsysteme:
    - Prozessverwaltung
    - Speicherverwaltung
      Virtuelles Dateisystem
    - Gerätetreiber

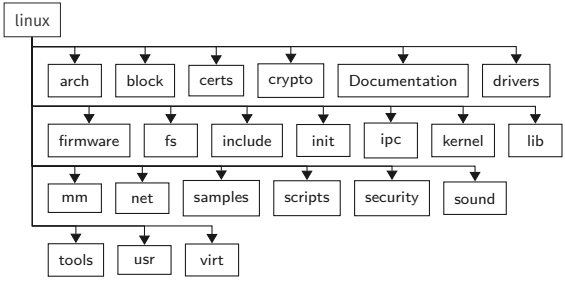
    ...

Linux Kernel

- **Eckdaten**
  - ursprünglich 1991 von Linus Torvalds entwickelt
  - veröffentlicht unter der GPLv2 [3]

- **Struktur** [4, 12, 14]
  - = dynamischer, monolithischer Kernel
  - dynamisch ladbare Module
  - (logische) Aufteilung in Subsysteme:
    - Prozessverwaltung
    - Speicherverwaltung
      Virtuelles Dateisystem
    - Gerätetreiber

    ...

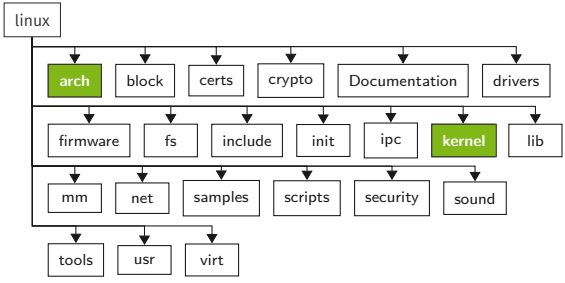


Linux Kernel

- **Eckdaten**
  - ursprünglich 1991 von Linus Torvalds entwickelt
  - veröffentlicht unter der GPLv2 [3]

- **Struktur** [4, 12, 14]
  - = dynamischer, monolithischer Kernel
  - dynamisch ladbare Module
  - (logische) Aufteilung in Subsysteme:
    - **Prozessverwaltung**
    - Speicherverwaltung
      Virtuelles Dateisystem
    - Gerätetreiber

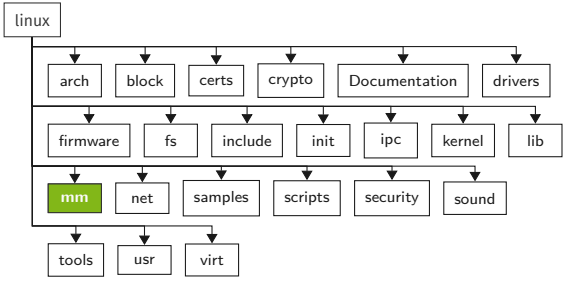
    ...



Linux Kernel

- **Eckdaten**
  - ursprünglich 1991 von Linus Torvalds entwickelt
  - veröffentlicht unter der GPLv2 [3]

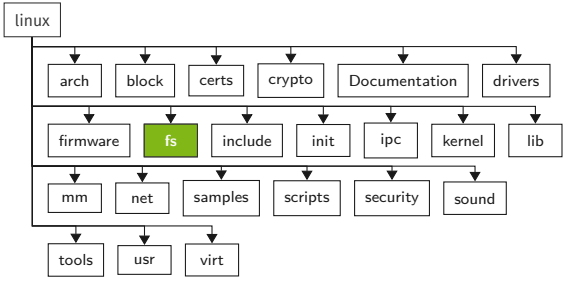
- **Struktur** [4, 12, 14]
  - = dynamischer, monolithischer Kernel
  - dynamisch ladbare Module
  - (logische) Aufteilung in Subsysteme:
    - Prozessverwaltung
    - **Speicherverwaltung**
    - Virtuelles Dateisystem
    - Gerätetreiber
  - ...



Linux Kernel

- **Eckdaten**
  - ursprünglich 1991 von Linus Torvalds entwickelt
  - veröffentlicht unter der GPLv2 [3]

- **Struktur** [4, 12, 14]
  - = dynamischer, monolithischer Kernel
  - dynamisch ladbare Module
  - (logische) Aufteilung in Subsysteme:
    - Prozessverwaltung
    - Speicherverwaltung
    - **Virtuelles Dateisystem**
    - Gerätetreiber
    - ...

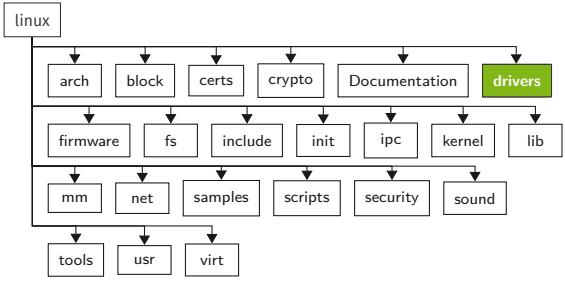


Linux Kernel

- **Eckdaten**
  - ursprünglich 1991 von Linus Torvalds entwickelt
  - veröffentlicht unter der GPLv2 [3]

- **Struktur** [4, 12, 14]
  - = dynamischer, monolithischer Kernel
  - dynamisch ladbare Module
  - (logische) Aufteilung in Subsysteme:
    - Prozessverwaltung
    - Speicherverwaltung
      Virtuelles Dateisystem
    - **Gerätetreiber**

    ...

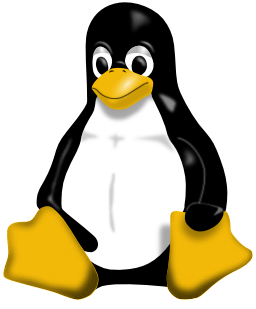


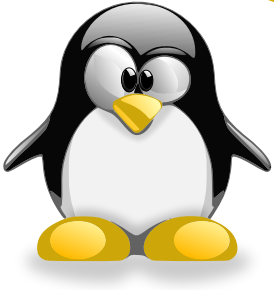
Agenda

Linux Entwicklungsprozess [5]

- **Community (Familie)**
  - große Entwicklungsgemeinschaft

    $\Rightarrow$ Privatpersonen, Wissenschaft, Unternehmen, ...
  - organisiert in einer Maintainer-Hierarchie
  - Kommunikation vorrangig per E-Mailverteiler (Linux Kernel Mailing List, LKML)





Linux Entwicklungsprozess [5]

- **Community (Familie)**
  - ...
- **Entwicklungsprozess**
  - neue stable Versionen alle 2-3 Monate
    - Veröffentlichung der stable Version *vX.W*

Linux Entwicklungsprozess [5]

- **Community (Familie)**
  - ...
- **Entwicklungsprozess**
  - neue stable Versionen alle 2-3 Monate
    - Veröffentlichung der stable Version *vX.W*
    - *Mergefenster* für nächste Version vX.Y

Linux Entwicklungsprozess [5]

- **Community (Familie)**
  - ...
- **Entwicklungsprozess**
  - neue stable Versionen alle 2-3 Monate
    - Veröffentlichung der stable Version *vX.W*
    - *Mergefenster* für nächste Version vX.Y
    - Stabilisierungsphase:
      1. *Release Candidate* (*vX.Y-rc1*)

Linux Entwicklungsprozess [5]

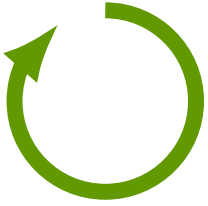
- **Community (Familie)**
  - ...
- **Entwicklungsprozess**
  - neue stable Versionen alle 2-3 Monate
    - Veröffentlichung der stable Version *vX.W*
    - *Mergefenster* für nächste Version vX.Y
    - Stabilisierungsphase:
      1. *Release Candidate* (*vX.Y-rc1*) ⇒ *vX.Y-rc2*

Linux Entwicklungsprozess [5]

- **Community (Familie)**
  - ...
- **Entwicklungsprozess**
  - neue stable Versionen alle 2-3 Monate
    - Veröffentlichung der stable Version *vX.W*
    - *Mergefenster* für nächste Version vX.Y
    - Stabilisierungsphase:
      1. *Release Candidate* (*vX.Y-rc1*) ⇒ *vX.Y-rc2* ⇒ *vX.Y-rc3*
      ⇒ ...

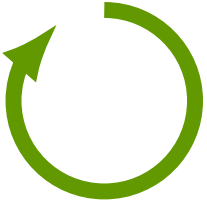
Linux Entwicklungsprozess [5]

- **Community (Familie)**
  - ...
- **Entwicklungsprozess**
  - neue stable Versionen alle 2-3 Monate
    - Veröffentlichung der stable Version *vX.W*
    - *Mergefenster* für nächste Version vX.Y
    - Stabilisierungsphase:
      1. *Release Candidate* (*vX.Y-rc1*) ⇒ *vX.Y-rc2* ⇒ *vX.Y-rc3*
      ⇒ ... ⇒ *Stable Release vx.Y*



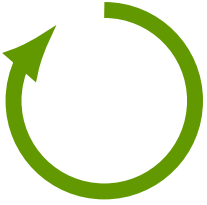
Linux Entwicklungsprozess [5]

- **Community (Familie)**
  - ...
- **Entwicklungsprozess**
  - neue stable Versionen alle 2-3 Monate
    - ...
  - kontinuierliche (Weiter-) Entwicklung der Funktionalität
    - Einreichung in Form von Patches
    - Diskussion per Linux Kernel Mailing List (*LKML*)
    - Nachbesserung der Patches
    - Aufnahme während *Mergefenster*



Linux Entwicklungsprozess [5]

- **Community (Familie)**
  - ...
- **Entwicklungsprozess**
  - neue stable Versionen alle 2-3 Monate
    - ...
  - kontinuierliche (Weiter-) Entwicklung der Funktionalität
    - Einreichung in Form von Patches
    - Diskussion per Linux Kernel Mailing List (*LKML*)
    - Nachbesserung der Patches
    - Aufnahme während *Mergefenster*
  - ⇒ **Umsetzung mittels Versionskontrolle**



Die einzelnen Schritte zum „eigenen“ Linux-Kern

Die einzelnen Schritte zum „eigenen“ Linux-Kern

1. Editor
2. Versionskontrolle
...

1. Quellcode beziehen
2. Konfiguration
3. Übersetzung

1. Doku lesen
2. Implementieren
3. Doku schreiben

1. Virtuelle Maschine
2. Testcases
3. Review

Editor

- **Editoren `mit` und `ohne` grafischer Oberfläche**
  - leichtgewichtig
  - einfach zu bedienen
  - Beispiele: vim, nano, geany, gedit, kate, ...

Editor

- **Editoren `mit` und `ohne` grafischer Oberfläche**
  - leichtgewichtig
  - einfach zu bedienen
  - Beispiele: vim, nano, geany, gedit, kate, ...
- **Entwicklungsumgebungen**
  - Schwergewichtig
  - bieten meist gute Code-Vervollständigung und Cross-Referenzierung
  - Beispiele: Eclipse, Visual Studio Code, ...

Editor

- **Editoren `mit` und `ohne` grafischer Oberfläche**
  - leichtgewichtig
  - einfach zu bedienen
  - Beispiele: vim, nano, geany, gedit, kate, ...
- **Entwicklungsumgebungen**
  - Schwergewichtig
  - bieten meist gute Code-Vervollständigung und Cross-Referenzierung
  - Beispiele: Eclipse, Visual Studio Code, ...

Es ist schwierig!
Bringt die Indezierung nicht selten an ihre Grenzen.

Editor

- **Editoren `mit` und `ohne` grafischer Oberfläche**
  - leichtgewichtig
  - einfach zu bedienen
  - Beispiele: vim, nano, geany, gedit, kate, ...
- **Entwicklungsumgebungen**
  - Schwergewichtig
  - bieten meist gute Code-Vervollständigung und Cross-Referenzierung
  - Beispiele: Eclipse, Visual Studio Code, ...

Es ist schwierig!
Bringt die Indezierung nicht selten an ihre Grenzen.

⇒ **Sucht Euch einen aus!** ☺

Versionskontrollsystem des Linux-Kernels git

- **Git**
  - **open source, verteiltes Versionskontrollsystem**
  - Sicherung von Daten und deren (alten) Zuständen
  - Zusammenführung paralleler Entwicklung
  - Transportmedium

Versionskontrollsystem des Linux-Kernels git

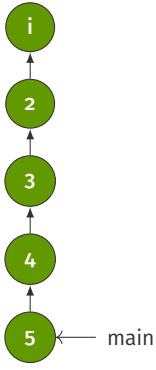
- **Git**
  - **open source, verteiltes Versionskontrollsystem**
  - Sicherung von Daten und deren (alten) Zuständen
  - Zusammenführung paralleler Entwicklung
  - Transportmedium
- **Prinzip**
  - Sammlung an Dateien in einem Repository

Versionskontrollsystem des Linux-Kernels git

- **Git**
  - **open source, verteiltes Versionskontrollsystem**
  - Sicherung von Daten und deren (alten) Zuständen
  - Zusammenführung paralleler Entwicklung
  - Transportmedium
- **Prinzip**
  - Sammlung an Dateien in einem Repository
  - Sicherung vollständiger Daten jeder Version (`commit`)

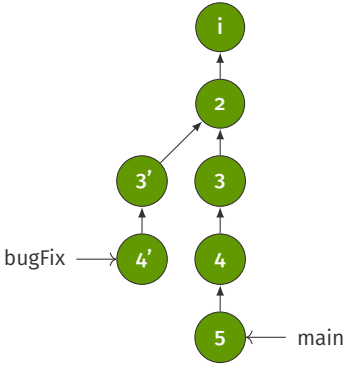
Versionskontrollsystem des Linux-Kernels git

- **Git**
  - **open source, verteiltes Versionskontrollsystem**
  - Sicherung von Daten und deren (alten) Zuständen
  - Zusammenführung paralleler Entwicklung
  - Transportmedium
- **Prinzip**
  - Sammlung an Dateien in einem Repository
  - Sicherung vollständiger Daten jeder Version (`commit`)
  - Jede Version kennt ihren Vorgänger (`parent`)



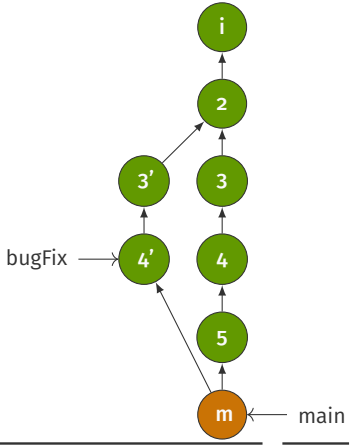
Versionskontrollsystem des Linux-Kernels git

- **Git**
  - **open source, verteiltes Versionskontrollsystem**
  - Sicherung von Daten und deren (alten) Zuständen
  - Zusammenführung paralleler Entwicklung
  - Transportmedium
- **Prinzip**
  - Sammlung an Dateien in einem Repository
  - Sicherung vollständiger Daten jeder Version (`commit`)
  - Jede Version kennt ihren Vorgänger (`parent`)
  - Parallele Entwicklung und Abzweigung von Versionsserien (`branches`)



Versionskontrollsystem des Linux-Kernels git

- **Git**
  - **open source, verteiltes Versionskontrollsystem**
  - Sicherung von Daten und deren (alten) Zuständen
  - Zusammenführung paralleler Entwicklung
  - Transportmedium
- **Prinzip**
  - Sammlung an Dateien in einem Repository
  - Sicherung vollständiger Daten jeder Version (`commit`)
  - Jede Version kennt ihren Vorgänger (`parent`)
  - Parallele Entwicklung und Abzweigung von Versionsserien (`branches`)
  - Zusammenführung von Versionsserien (`merge`)



Versionskontrollsystem des Linux-Kernels – Hilfestellungen

- Interaktives Git-Tutorial:
  `https://learngitbranching.js.org/`
- Eigene Einführung in Git:
  `https://sys-sideshow.cs.tu-dortmund.de/lehre/ws23/bsb/folien/seminar-git.pdf`
  Achtung: Die Einführung ist für die Veranstaltung „Betriebssystembau“ gedacht. Daher stimmen URLs und manche Szenarios nicht.

Die einzelnen Schritte zum „eigenen“ Linux-Kern

Die einzelnen Schritte zum „eigenen“ Linux-Kern

Die einzelnen Schritte zum „eigenen“ Linux-Kern

Die Linux-Kern-Konfiguration – Überblick

- **Der Linux-Kernel unterstützt …**
  - 20 CPU Architekturen
  - 70 Dateisysteme
  - 40 Ethernet Gerätetreiber
  - …

Die Linux-Kern-Konfiguration – Überblick

- **Der Linux-Kernel unterstützt …**
  - 20 CPU Architekturen
  - 70 Dateisysteme
  - 40 Ethernet Gerätetreiber
  - …

Achtung

nur ein **Bruchteil** der gesamten Codebasis wird tatsächlich pro System benötigt

Die Linux-Kern-Konfiguration – Überblick

- **Der Linux-Kernel unterstützt …**
  - 20 CPU Architekturen
  - 70 Dateisysteme
  - 40 Ethernet Gerätetreiber
  - …

Wichtig

Kernel auf Bedürfnisse, Hardware, Komponenten, … zuschneiden

⇒ **Kernel-Konfiguration**

Die Linux-Kern-Konfiguration – Überblick

- **Der Linux-Kernel unterstützt …**
  - 20 CPU Architekturen
  - 70 Dateisysteme
  - 40 Ethernet Gerätetreiber
  - …

Wichtig

Kernel auf Bedürfnisse, Hardware, Komponenten, … zuschneiden
⇒ **Kernel-Konfiguration**

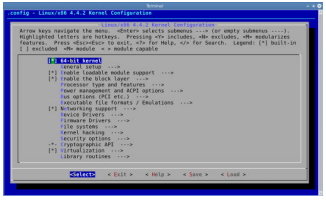
- **Kconfig**
  - = Konfigurationssprache
  - verwaltet alle Compile-Zeit Konfigurationsoptionen
  - Berücksichtigt Abhängigkeiten und setzt diese durch
  - Erstellung der Konfiguration (`.config`) durch Tools wie `menuconfig`

⚠ Konfigurationen nicht über verschiedene Kernel Versionen hinweg stabil

Die Linux-Kernel Konfiguration – Aufbau [6, 7]

`menuconfig`

- = menu-basiertes Konfigurationsinterface
- kurze Beschreibung pro Option
- Sicherstellung von Abhängigkeiten
- Suchfunktionaltiät



Bedienung

- ? Beschreibung / Tipp anzeigen
- esc abbrechen / beenden
- / Suche

Symbole

- < > keine Abhängigkeiten
- [ ] kann einkompiliert (y) werden oder nicht (n)
- { } als Modul (m) oder einkompiliert (y) benötigt
- -- einkompiliert (y) benötigt

Linux-Kernel-Makefiles [8]

- **Makefile**
  - = beschreibt Beziehung zwischen Dateien; definiert Kommandos zum Updaten von Dateien
  - ⇒ Grundlage für die Nutzung von GNU *make*

Linux-Kernel-Makefiles [8]

- **Makefile**
  - = beschreibt Beziehung zwischen Dateien; definiert Kommandos zum Updaten von Dateien
  - ⇒ Grundlage für die Nutzung von GNU *make*

Linux Kernel Makefiles

```
linux
├── Makefile
├── .config
├── arch
│   └── $(SRCARCH)/Makefile
├── scripts
│   └── Makefile.*
└── */Makefile (kbuild)
```

Linux-Kernel-Makefiles [8]

- **Makefile**
  - = beschreibt Beziehung zwischen Dateien; definiert Kommandos zum Updaten von Dateien
  - ⇒ Grundlage für die Nutzung von GNU *make*

Linux Kernel Makefiles

```
linux
└── Makefile
```

Hilfreiche Targets

menuconfig	–	Edit config with menu-based program
bzImage	–	Build compressed kernel image
vmlinux	–	Build bare kernel
modules	–	Build all modules

Die einzelnen Schritte zum „eigenen“ Linux-Kern

Die einzelnen Schritte zum „eigenen“ Linux-Kern

Die einzelnen Schritte zum „eigenen“ Linux-Kern

Linux in der Virtuellen Maschine (VM)

- **Virtualisierung**
  - = Abstraktionsschicht zwischen Hardware und Anwendung(en)

Linux in der Virtuellen Maschine (VM)

- **Virtualisierung**
  - = Abstraktionsschicht zwischen Hardware und Anwendung(en)

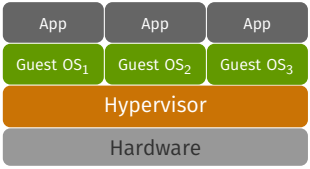
- **Emulation**
  - = funktionelles Nachbilden eines System durch ein anderes

Linux in der Virtuellen Maschine (VM)

- **Virtualisierung**
  - = Abstraktionsschicht zwischen Hardware und Anwendung(en)

- **Vollständige Virtualisierung** [15] [17, Seite 627 ff.]
  - = Simulation der Hardware (transparente Virtualisierung)
  - Hypervisor als Virtualisierungsmanager
  - ① *bare-metal* oder ② *hosted*

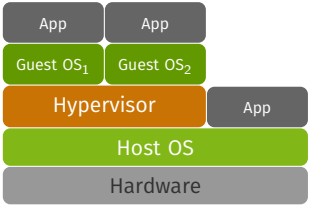
Linux in der Virtuellen Maschine (VM)



- **Virtualisierung**
  = Abstraktionsschicht zwischen Hardware und Anwendung(en)
- **Vollständige Virtualisierung** [15] [17, Seite 627 ff.]
  = Simulation der Hardware (transparente Virtualisierung)
  - Hypervisor als Virtualisierungsmanager
  - ① ***bare-metal*** oder ② *hosted*

Linux in der Virtuellen Maschine (VM)

- **Virtualisierung**
  = Abstraktionsschicht zwischen Hardware und Anwendung(en)

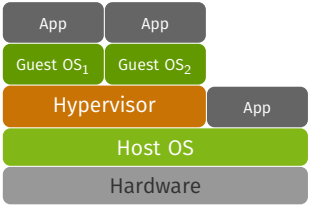


② *hosted*

- **Vollständige Virtualisierung** [15] [17, Seite 627 ff.]
  = Simulation der Hardware (transparente Virtualisierung)
  - Hypervisor als Virtualisierungsmanager
  - ① *bare-metal* oder ② ***hosted***

Linux in der Virtuellen Maschine (VM)

- **Virtualisierung**
  - = Abstraktionsschicht zwischen Hardware und Anwendung(en)



② *hosted*

- **Vollständige Virtualisierung** [15] [17, Seite 627 ff.]
  - = Simulation der Hardware (transparente Virtualisierung)
  - Hypervisor als Virtualisierungsmanager
  - ① *bare-metal* oder ② *hosted*
- **Paravirtualisierung**
  - = Variante der vollst. Virtualisierung
  - spezielle Hypervisor-Schnittstelle fuer bessere Performance
  - erfordert OS-Anpassungen

QEMU/KVM (1/2)

QEMU [9]

- = freier, open source Emulator
- ⇒ erstellt das virtuelle System (VM), inkl. CPU, Speicher und Geräte

KVM [10]

- = Virtualisierungsmodul im Linux Kernel
- ⇒ Schnittstelle zwischen VM und Hardware
- ⇒ ermöglicht Paravirtualisierung

Nutzung in Kombination:

```
01 stud@sst:~ qemu-system-x86_64 -enable-kvm [options] [disk_image]
```

Bedienung

Kommandos werden mit Escape Sequenz eingeleitet: Ctrl + a

- h Hilfe anzeigen
- x Emulation beenden
- t Konsolenzeitstempel umschalten
- b Abbruch senden (Magic SysRq)

QEMU/KVM (2/3)

① Virtuelle Festplatte einrichten [11]

- Datei in der Festplatteninhalte der VM gespeichert werden
- Speicherplatz Optimierung durch `qcow` Dateiformat

```
01 stud@sst:~ qemu-img create -f qcow2 FILENAME.img [SIZE]
```

② Distribution installieren [11]

- Minimaler Debian-Installer: Kernel + Installationprogramm (Ramdisk)
- Nutzung des Debian Spiegels zum Nachladen weiterer Programme

```
01 stud@sst:~ qemu-system-x86_64 -drive file=FILENAME.img <...> \
02                   -cdrom INSTALLER.iso
```

QEMU/KVM (3/3)

③ **Booten des Gastsystems** [11]

- Booten ohne Ramdisk
- Nutzung des eigenen Kernels

```
01 stud@sst:~ qemu-system-x86_64 -drive file=FILENAME.img <...> -kernel CUSTOM_KERNEL
```

QEMU/KVM (3/3)

③ **Booten des Gastsystems** [11]

- Booten ohne Ramdisk
- Nutzung des eigenen Kernels

```
01 stud@sst:~ qemu-system-x86_64 -drive file=FILENAME.img <...> -kernel CUSTOM_KERNEL
```

Hinweise für unser Setup [10, Siehe `virtio`]

- `boot.sh` Skript mit passenden `QEMU` Flags unter `/fs/proj/sst/`
- Paravirtualisierung mittels Virtio ⇒ Kernel benötigt Virtio Treiber
- Freigabe von Verzeichnissen zwischen Host und Gast per *Plan 9 Folder Sharing* (Siehe Aufgabezettel)

Agenda

Der Linux-Kern

- **Ziel:**
  - Debian VM mit selbstkonfiguriertem Kernel
  - Dateienaustausch zwischen Host-System und VM
  - Möglichst kleine Kernelkonfiguration (*Challenge*)

Der Linux-Kern

- **Ziel:**
  - Debian VM mit selbstkonfiguriertem Kernel
  - Dateienaustausch zwischen Host-System und VM
  - Möglichst kleine Kernelkonfiguration (*Challenge*)
- **Hinweise:**
  - Kernelquellen sind ein **sehr** großes Projekt
    $\Rightarrow$ nur Tag `v6.1.55` ohne Historie klonen (siehe `git clone` Flags)

Der Linux-Kern

- **Ziel:**
  - Debian VM mit selbstkonfiguriertem Kernel
  - Dateienaustausch zwischen Host-System und VM
  - Möglichst kleine Kernelkonfiguration (*Challenge*)
- **Hinweise:**
  - Kernelquellen sind ein **sehr** großes Projekt

    $\Rightarrow$ nur Tag `v6.1.55` ohne Historie klonen (siehe `git clone` Flags)
  - Ablage & Bauen des Linuxkerns lokal unter `/fs/scratch`

    kein Backup per NFS, aber schneller

Der Linux-Kern

- **Ziel:**
  - Debian VM mit selbstkonfiguriertem Kernel
  - Dateienaustausch zwischen Host-System und VM
  - Möglichst kleine Kernelkonfiguration (*Challenge*)
- **Hinweise:**
  - Kernelquellen sind ein **sehr** großes Projekt
    ⇒ nur Tag `v6.1.55` ohne Historie klonen (siehe `git clone` Flags)
  - Ablage & Bauen des Linuxkerns lokal unter `/fs/scratch`
    ⚠ **kein Backup per NFS, aber schneller**
  - Kernelkonfiguration: Anforderungen an Kernel beachten
    (Dateienaustausch per 9p FS, Virtio Paravirtualisierung, ...)

Der Linux-Kern

- **Ziel:**
  - Debian VM mit selbstkonfiguriertem Kernel
  - Dateienaustausch zwischen Host-System und VM
  - Möglichst kleine Kernelkonfiguration (*Challenge*)
- **Hinweise:**
  - Kernelquellen sind ein **sehr** großes Projekt
    ⇒ nur Tag `v6.1.55` ohne Historie klonen (siehe `git clone` Flags)
  - Ablage & Bauen des Linuxkerns lokal unter `/fs/scratch`
    ⚠ **kein Backup per NFS, aber schneller**
  - Kernelkonfiguration: Anforderungen an Kernel beachten
    (Dateienaustausch per 9p FS, Virtio Paravirtualisierung, ...)
  - Abgabe der Kernelconfig: **nur** die `.config` Datei einchecken!
    ⚠ **nicht die gesamten Kernelquellen committen!**

Agenda

Referenzen

[1] URL: https://gs.statcounter.com/os-market-share/mobile/worldwide.

[2] URL: https://w3techs.com/technologies/overview/operating_system.

[3] URL: https://www.gnu.org/licenses/old-licenses/gpl-2.0.html.

[4] URL: https://linux-kernel-labs.github.io/refs/heads/master/lectures/intro.html#linux-source-code-layout.

[5] URL: https://www.kernel.org/doc/html/latest/process/2.Process.html#the-big-picture.

[6] URL: https://www.kernel.org/doc/html/latest/kbuild/kconfig-language.html.

[7] URL: https://opensource.com/article/18/10/kbuild-and-kconfig.

[8] URL: https://www.kernel.org/doc/html/latest/kbuild/makefiles.html.

[9] URL: https://www.qemu.org/docs/master/index.html.

[10] URL: https://www.linux-kvm.org/.

[11] URL: https://wiki.archlinux.org/title/QEMU.

[12] Daniel Pierre Bovet und Marco Cesati. **Understanding The Linux Kernel.** 3rd. O'Reilly Media Inc., Nov. 2005. ISBN: 0596005652.

[13] Jonathan Corbet, Alessandro Rubini und Greg Kroah-Hartman. **Linux Device Drivers, 3rd Edition.** O'Reilly Media, Inc., 2005. ISBN: 0596005903.

[14] Robert Love. **Linux Kernel Development.** 3rd. Juni 2010. ISBN: 9780672329463.

[15] Karenb Scarfone, Murugiah Souppaya und Paul Hoffman. **Guide to Security for Full Virtualization Technologies.** Techn. Ber. National Institute of Standards und Technology, Jan. 2011. URL: https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-125.pdf.

[16] Abraham Silberschatz, Peter Baer Galvin und Greg Gagne. **Operating System Concepts.** 10th. Wiley, 2018. ISBN: 978-1-118-06333-0.

[17] William Stallings. **Operating Systems: Internals and Design Principles.** 9th. Pearson, 2017. ISBN: 9781292214306; 9781292214290.

[18] Wired. Aug. 2016. URL: https://www.wired.com/2016/08/linux-took-web-now-taking-world/.